Small Algorithms for Small Systems

James H. Davenport
Department of Computer Science, University of Bath
Bath BA2 7AY, United Kingdom
J.H.Davenport@bath.ac.uk

Knuth is said to have described Computer Science as "that part of mathematics in which $\log\log n = 3$". This paper only considers some parts of Computer Algebra, and the even more special case when $\log n \approx 3$, or even less, and where compactness of the algorithm itself, as well as the data structures, is important.

We begin with a few remarks on integration, which, while in some sense the culmination of computer algebra for the "compact computer algebra" market, also inspired many of our other suggestions.

Acknowledgements. This paper was prepared when the author was visiting the Symbolic Computation Group in the David R. Cheriton School of Computer Science, University of Waterloo, and the author is grateful to Prof. Giesbrecht and colleagues for their hospitality. Its preparation was inspired by an invitation to talk at the Compact Computer Algebra workshop at CICM 2009, and the author is grateful to Dr Smirnova for the invitation. At that workshop Prof. Geddes remarked that the subresultant algorithm did not work well for multivariates, which inspired section 5.

1 Integration

Here the Risch–Norman [NM77] (also called 'parallel Risch' [Bro07]) algorithm can be quite short to program, and, while not a full decision procedure, *is* complete on a reasonable range of transcendental integrands [Dav82, Bro07]. There is a recent extension [Kau08], which looks promising on many cases of algebraic integrands. Here the aim would be to integrate *correctly* many common cases, while *not* guaranteeing that "I can't" is equivalent to "no-one can". We should note that, while the traditional "full Risch" integration algorithm is adapted to a recursive representation of polynomials, Risch–Norman is equally adapted to a distributed representation (where in fact it was first implemented [NM77]). Since this algorithm is the inspiration for algorithms 3–5 below, and since we claim that there is great commonality of technology, we give an appropriate sketch here.

Algorithm 1 (Risch–Norman Method [NM77]) *To integrate $f(x, \theta_1, \ldots, \theta_n)$, where f is a rational function with coefficients in some field C of constants (of characteristic zero), and each θ_i is defined over $K_i = C(x, \theta_1, \ldots, \theta_{i-1})$ by one of:*

log $\theta_i = \log u_i$, *i.e.* $\theta_i' = \frac{u_i'}{u_i}$, *where $u_i \in K_i$ (in practice we impose $u_i \in C[x, \theta_1, \ldots, \theta_{i-1}]$);*

exp $\theta_i = \exp u_i$, *i.e.* $\theta_i' = u_i'\theta_i$;

sin $\theta_i = \sin u_i$, *i.e.* $\theta_i' = u_i' \cos u_i$;

cos $\theta_i = \cos u_i$, *i.e.* $\theta_i' = u_i' \sin u_i$;

These two don't quite satisfy "θ_i is defined over the previous", but as long as we introduce them in adjacent pairs, i.e. if $\theta_i = \sin u_i$, one of $\theta_{i\pm1} = \cos u_i$ and *vice versa*, this seems to work in practice. Alternatively, we can employ the half-angle substitution, and use

tan $\theta_i = \tan u_i$, *i.e.* $\theta_i' = u_i'(1 + \theta_i^2)$;

arctan $\theta_i = \arctan u_i$, *i.e.* $\theta_i' = \frac{u_i'}{1+u_i^2}$, *where* $u_i \in K_i$.

1. Perform a certain amount of normalization on f. In principle we ought to do a full application of the Risch Structure Theorem [Ris79], but in practice we do rather less, e.g. replacing $\exp(2x)$ by $\exp(x)^2$. [Dav86] suggests some additional normalizations, which, while not strictly necessary according to [Ris79], reduce (see question 1 below) the incidence of pathological counter-examples to the completeness of the Risch–Norman Method.

2. Write $f = p/q$ where $p, q \in C[x, \theta_1, \ldots, \theta_n]$. It may be necessary to introduce apparently spurious common factors, e.g. a $\frac{1}{x}$ if the integrand involves $\log x$, as described in step 7.

3. Compute $r = \gcd(q, \mathrm{num}(q'))$ and $s = \gcd(r, \mathrm{num}(r'))$, where $'$ denotes *total* derivative, and the num deals with any denominators that come from differentiating log, arctan etc.

4. Produce a list $\mathcal{L}$ of "new functions". Conceptually we ought to factor q over the algebraic closure $\overline{C}$, and for each factor f, add $\log f$ to $\mathcal{L}$. In practice we tend to take a short-cut, and compute either a factorization over C, or even just a square-free decomposition, and, for each factor f

   (a) add $\log f$ to $\mathcal{L}$;

   (b) if f is a quadratic $p\alpha^2 + q\alpha + r$, where α is x, any of the θ_i, or a power of any of these, *and* if $4pr - q^2$ is either a square[1] or a positive constant, add $\arctan\left(\frac{2p\alpha+q}{\sqrt{4pr-q^2}}\right)$ to $\mathcal{L}$ (and if p itself is not constant, add $\log p$ to $\mathcal{L}$ as well);

   (c) if $4pr - q^2$ is minus a square[1], or a negative constant, replace $\log f$ by its summands $\log\left(2p\alpha + q \pm \sqrt{q^2 - 4pr}\right)$ (and if p itself is not constant, add $\log p$ to $\mathcal{L}$ as well);

   (the case where $4pr - q^2$ is neither $\pm$ a square[1] nor a constant does *not* lead to the addition of new terms, by Liouville's principle)

   (d) If f is univariate (of degree greater than 2) in α (being x or any of the θ_i), replace $\log f$ by its summands $\log(\alpha - \texttt{RootOf}(f, \alpha, i))$ with i from 1 to $\deg_\alpha(f)$.

   This last case requires a sophisticated `RootOf` construct, and the author suspects is unlikely to be implemented in a compact system. Note that, if we have done a complete factorization in this step, we only need this case to express integrals such as

$$\int \frac{1}{x^5 + x + 2}\mathrm{d}x = \frac{1}{6}\log(x+1) + \sum_{\beta = \texttt{RootOf}(8376\gamma^4 + 1396\gamma^3 + 206\alpha^2 + 21\alpha + 1)} \beta \log\left(10\beta + x(1 + 4\beta)\right).$$

[1] Up to multiplying by a positive constant.

5. Conceptually, express the integral as

$$I := \frac{\sum a_{i,j,k,\ldots} x^i \theta_1^j \theta_2^k \ldots}{r} + \sum_{f_i \in \mathcal{L}} c_i f_i. \tag{1}$$

6. Clear denominators in $\frac{p}{q} = I'$ to get

$$p - \sum_{f_i \in \mathcal{L}} c_i f_i' q = \sum a_{i,j,k,\ldots} x^i \theta_1^j \theta_2^k \left(\frac{r'}{s} + \frac{q}{r} \underbrace{\left(\frac{i}{x} + \frac{j\theta_1'}{\theta_1} + \frac{k\theta_2'}{\theta_2} + \cdots \right)}_{\text{see note 7}} \right) \tag{2}$$

7. The divisions in $\frac{r'}{s}$ and $\frac{q}{r}$ should be exact with respect to the leading variables, but there may be factors corresponding to the num operators in step 3. Similarly, the explicit divisions by x or θ_i in (2) are exact, since x or θ_i appears to a non-zero power in the multiplicand, but there may be denominators of θ_i' which have to be allowed for in the denormalization in step 2.

8. Generate and solve, iteratively, the linear equations for the $a_{i,j,k,\ldots}$, starting with the leading monomial on the left-hand side of (2). For the details of this, see [NM77].

At some points, we will determine the c_i, and, if we have a contradiction, we see that the integral is *definitely* not in the form (1), and therefore *probably* not integrable in elementary terms (again, see question 1).

2 g.c.d. of univariate polynomials

This has been a bugbear of computer algebra for over forty years, and has given rise to many solutions, some of them truly heroic (see [CGG84, DP85], where the then 'compact kernel' of Maple did not extend as far as polynomials with modular coefficients, so the primitive parts of univariate g.c.d.s were computed via the diagram

$$\begin{array}{rccc} \mathbb{Z}[x] & \overset{\text{makeprim}}{\longrightarrow} & \overset{\text{gcd}}{\longrightarrow} & \mathbb{Z}[x] \\ \text{x:=N} \Downarrow & & & \Uparrow \text{N:=x} \\ \mathbb{Z} & \overset{\text{gcd}}{\Longrightarrow} & \overset{\text{makeprim}}{\Longrightarrow} & \mathbb{Z} \end{array}$$

where 'makeprim' is the operation of making a polynomial primitive — content removal).
Though difficult to *prove*, the subresultant algorithm [Col67] is quite short to *program*, and its intermediate expression swell does not manifest itself on small examples. It may well be worth considering the trial division variant of [Hea79]. However, for multivariate examples, we may wish to be more subtle — see below.

3 Factoring of univariate polynomials

This has been a challenge for almost as long as the g.c.d. problem, and is still far from being solved, as significant improvements keep on being made (see [vH02] for one of the more recent major developments). Nevertheless, if $\log \log n$ is small, we can devise a relatively simple algorithm on the following lines.

Algorithm 2 *To factor a square-free $f(x)$ over $\mathbb{Z}$ ($\mathbb{Q}$ is equivalent).*

1. To factor mod p, use Cantor–Zassenhaus [CZ81], after checking that f remains square-free when reduced modulo p.
2. Possibly use several primes — see question 2 below.
3. Having decided that there is a viable factorization, we have to lift it p-adically. Again, we note that while an optimal lifting is a very complicated body of code, linear lifting [DST93, p. 168], with imposed leading coefficients [DST93, pp. 174–5], is not, and for small polynomials the difference in performance is not marked.
4. Obviously, any p-adic factor which divides over the integers is a true factor. If this doesn't happen, we have two choices.
   (a) Do appropriate recombinations and trial divisions. The code is not lengthy, but the running may well be, since most optimizations ([ABD85] is possibly a counterexample) will substantially lengthen the code.
   (b) Just give up, and declare "I couldn't find any factors, but they may nonetheless exist". In practice, this may well be acceptable on a compact system.

4 Factoring of multivariate polynomials

It's not clear to this author that this is worth implementing. However, if one does, the following "low brow" algorithm may be worth considering: essentially linear lifting with imposed leading coefficients. We describe it here in the bivariate case: see section 6 for the general case. Note that we assume the input is square-free, which implies that we need the g.c.d. of multivariate polynomials — see the following section.

Algorithm 3 *To factor a square-free $f(x, y)$ over $\mathbb{Z}$.*

1. Ensure $f(x, 0)$ is square-free, and of the same degree in x as $f(x, y)$. If not, make a substitution $y \mapsto y + v$ for small integer v until it is.
2. Factor $f(x, 0)$ as a product of k factors.
3. Multiply each such factor h by $\operatorname{lc}_x f(x, 0)/\operatorname{lc}_x(h)$, so that we have a factorization of
$$\left(\operatorname{lc}_x f(x, 0)\right)^{k-1} f(x, 0) = \prod_{i=1}^{k} g_i,$$
where each g_i has leading coefficient $\operatorname{lc}_x f(x, 0)$. Replace f by $\left(\operatorname{lc}_x f(x, y)\right)^{k-1} f(x, y)$.
4. Replace $g_i = \sum_j a_{i,j} x^j$ by $h_i = \sum_j (a_{i,j} + b_{i,j} y)\, x^j$ where the $b_{i,j}$ are unknowns, save that the leading coefficient of g_i has cy added, where c is the coefficient of y^1 in $\operatorname{lc}_x f$.
5. Equate to zero the coefficients (with respect to x) of $\left.\frac{f(x,y) - \prod g_i}{y}\right|_{y=0}$.

The leading coefficient need not be computed, since it is forced to be correct by step 3

6. Solve this (linear!) system for the $b_{i,j}$.
7. Replace $g_i = \sum_j (a_{i,j} + b_{i,j} y) x^j$ by $h_i = \sum_j \left(a_{i,j} + b_{i,j} y + c_{i,j} y^2\right) x^j$ where the $c_{i,j}$ are unknowns, save that the leading coefficient of g_i has cy^2 added, where c is the coefficient of y^2 in $\mathrm{lc}_x f$.
8. Equate to zero the coefficients (with respect to x) of $\left.\frac{f(x,y) - \prod g_i}{y^2}\right|_{y=0}$.
9. Solve this (linear!) system for the $c_{i,j}$.
10. Repeat steps 7–9 to find the coefficients of $y^3, y^4, \ldots$ until one of two things happens.
    (a) $f(x,y) - \prod g_i = 0$, when we have a complete factorization.
    (b) The computation of $\prod g_i$ starts generating terms in y^k where $k > \deg_y f$, which shows that our univariate factorization was bad, in the sense that $f(x,0)$ factors more than $f(x,y)$.
11. Remove contents from each of the factors, so that we have a factorization of the original f, rather than the adjusted version from step 3.

In case 10b, we have two choices, as at the end of section 3.

1. Do appropriate recombinations and trial divisions. The code is not particularly lengthy, but the running may well be, since most optimizations are quite complicated. See also question 3 below.
2. Just give up, and declare "I couldn't find any factors, but they may nonetheless exist". In practice, this may well be acceptable on a compact system.

5 g.c.d. of bivariate polynomials

The subresultant algorithm [Col67] is, as we have said, not optimal for univariates, but in practice is probably good enough. For multivariates, the expression swell is intolerable, and we need a better algorithm. We can use the same process as algorithm 3 for the bivariate case: see section 6 for the general case.

Algorithm 4 *To compute* $\gcd(f_1, f_2)$ *in* $\mathbb{Z}[x,y]$.

1. Choose a value v for y, checking that substituting this value does not cause the x-degrees of f_1 or f_2 to drop.
2. Compute $g_1 = \gcd(f_1(x,v), f_2(x,v))$.
3. **If** $\gcd(g_1, f_1(x,v)/g_1) = 1$, then $h := f_1$, $g_2 := f_1(x,v)/g_1$.
4. **Else If** $\gcd(g_1, f_2(x,v)/g_1) = 1$, then $h := f_2$, $g_2 := f_2(x,v)/g_1$.

5. **Else** pick random λ, μ, check $\gcd(g_1, (\lambda f_1(x,v) + \mu f_2(x,v))/g_1) = 1$, then $h := \lambda f_1 + \mu f_2$, $g_2 := (\lambda f_1(x,v) + \mu f_2(x,v))/g_1$.

6. Lift the factorization $h(x,v) = g_1 g_2$ to a factorization of h, at which point the factor corresponding to g_1 is the required g.c.d.

7. Failure to lift, which is detected by the fact that $g_1 g_2$ generates powers of y greater than $\deg_y h$, implies that v was unlucky, and we go back to step 1.

An "early termination" test is possible in step 6, but may not be worth it in our context.

6 Multivariate lifting

The lifting processes involved in sections 4 and 5 could be described as "poor man's Hensel's Lemma". The normal process of lifting a factorization, whether for its own sake or as a g.c.d. computation, would be to lift the coefficients (with respect to x) of our polynomial from being in $k[y]$ to being in $k[y,z]$. This would generally be quite a substantial piece of code. It is worth noting that [MN81] described their multivariate g.c.d./factorization package as being "at least as large" as their (fairly complete) implementation of the Risch–Norman integration method.
Inspired by the non-recursive nature of the Risch–Norman method, we suggest an alternative. Rather than engage in lifting of *polynomials*, we continue with lifting *coefficients*.

Algorithm 5 *Continuation of algorithm 3, to factor $f(x,y,z)$. Assume we have factored $f(x,y,0) = \prod g_i(x,y)$.*

11. For each i, if $\deg_x(g_i) = d_i$, add $cx^{d_i}z + \sum_{j=0}^{d_i-1} c_{i,j}^{(0)} x^j z$ to g_i, where c is the coefficient of zy^0 in $\mathrm{lc}_x(f)$, and the $c_{i,j}^{(0)}$ are unknowns.

12. Equate to zero the coefficients (with respect to x) of $\left.\frac{f(x,y,z)-\prod g_i}{z}\right|_{z,y=0}$.

    The leading coefficient need not be computed, since it is forced to be correct by the term $cx^{d_i}z$ in the previous step.

13. Solve this (linear!) system for the $c_{i,j}^{(0)}$.

14. For each i, if $\deg_x(g_i) = d_i$, add $cx^{d_i}zy + \sum_{j=0}^{d_i-1} c_{i,j}^{(1)} x^j zy$ to g_i, where c is the coefficient of zy^1 in $\mathrm{lc}_x(f)$, and the $c_{i,j}^{(1)}$ are unknowns.

15. Equate to zero the coefficients (with respect to x) of $\left.\frac{f(x,y,z)-\prod g_i}{yz}\right|_{z,y=0}$.

16. Solve this (linear!) system for the $c_{i,j}^{(1)}$.

17. Repeat steps 14–16 for the coefficients of $zy^2, zy^3, \ldots$, until the termination criteria as in steps 10a, 10b happen, then $z^2y^0, z^2y, \ldots$, then higher powers of z, again using the termination criteria as in steps 10a, 10b.

18. Repeat for subsequent variables.

Note that we are using "imposed leading coefficients", which is recognised to be expensive. [Wan78] has suggestions for avoiding this, but the coding effort (not least arranging for recursive factorization of the leading coefficient) is substantial. It is not clear that the effort is justified, especially as one major application of this lifting will be to g.c.d. computations, where there are only two factors, so the blow-up is less significant. See also question 4 below.

Open Research Questions

Despite the "not cutting edge" nature of the algorithms being considered, we can see a few open questions.

Question 1 *Assuming we do the further normalizations described in [Dav86], and assuming we have done a full application of the Risch Structure Theorem [Ris79], under what circumstances does the Risch–Norman method (algorithm 1) fail to return an integral?*

Put another way, we need to attach a 'health warning' to algorithm 1 on the following lines, and the question above asks how to complete it:

> If the 'integrate' button returns an unevaluated integral expression, this normally means that no elementary (this expression has a technical definition — see e.g. [DST93, p. 191]) expression *can be* the desired integral: the exceptions are cases where the system has not detected hidden dependencies in the integral [i.e. not applied the full Risch Structure Theorem], or ...

Question 2 *How many primes p_i should we factorize modulo in step 2 above before deciding that we have a compatible factorization, and should proceed to Hensel lifting.*

[Mus78] suggests that the answer is 5, though there are heuristic arguments that this should grow as $\log \log d$, where d is the degree of the polynomial to be factored. If d is small, can we get away with less?

Question 3 *For recombinations of bivariate (or indeed multivariate) polynomial factorizations, are there equivalent simple optimizations to those of [ABD85] in the univariate case?*

It seems likely that there are, though the details will probably depend critically on the implementation.

Question 4 *In the g.c.d. case of lifting (algorithms 4, 5), can we do better than "imposed leading coefficients"?*

At least in principle, the answer is affirmative. In the notation of algorithm 4, step 6, we are trying to lift the factorization $h(x, \mathbf{v}) = g_1 g_2$ to a factorization of $h = \lambda f + \mu g$, where the factor corresponding to g_1 is the required g.c.d. of the original f and g. Since

$$\mathrm{lc}_x \gcd(f, g) | \gcd(\mathrm{lc}_x f, \mathrm{lc}_x g) = H$$

it suffices to impose leading coefficients of H on g_1 and $\mathrm{lc}_x\, h$ on g_2, thus effectively lifting a factorization of Hh rather than $\mathrm{lc}_x(h)h$.
Whether it is worth it in practice is another question. We note the complexity (rather than the actual cost, which will almost certainly be recouped on average) of a recursive g.c.d. computation, and the fact that for square-free decomposition, the main application of g.c.d. computation in integration, there is no gain.

Conclusion

Although the optimal versions of modern computer algebra algorithms are pretty lengthy and complicated objects, there are surprisingly compact 'low brow' versions of them which can be remarkably effective on small examples. Even these low brow versions throw up some interesting research questions, as in the previous section.
Furthermore, the underlying technology base, which, apart from modular arithmetic to support Cantor–Zassenhaus and *univariate* Hensel, consists of polynomial manipulation and linear equation solving, is fairly small and common across several of the higher-level algorithms.

References

[ABD85] J.A. Abbott, R.J. Bradford, and J.H. Davenport. A Remark on Factorisation. *SIGSAM Bulletin 2*, 19:31–33, 1985.

[Bro07] M. Bronstein. Structure theorems for parallel integration (Paper completed by Manuel Kauers). *J. Symbolic Comp.*, 42:757–769, 2007.

[CGG84] B.W. Char, K.O. Geddes, and G.H. Gonnet. GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation. In J.P. Fitch, editor, *Proceedings EUROSAM 84*, pages 285–296, 1984.

[Col67] G.E. Collins. Subresultants and Reduced Polynomial Remainder Sequences. *J. ACM*, 14:128–142, 1967.

[CZ81] D.G. Cantor and H. Zassenhaus. A New Algorithm for Factoring Polynomials over Finite Fields. *Math. Comp.*, 36:587–592, 1981.

[Dav82] J.H. Davenport. On the Parallel Risch Algorithm (I). In *Proceedings EUROCAM '82 [Springer Lecture Notes in Computer Science 144*, pages 144–157, 1982.

[Dav86] J.H. Davenport. On the Risch Differential Equation Problem. *SIAM J. Comp.*, 15:903–918, 1986.

[DP85] J.H. Davenport and J.A. Padget. HEUGCD: How Elementary Upperbounds Generate Cheaper Data. In *Proceedings EUROCAL 85*, pages 18–28, 1985.

[DST93] J.H. Davenport, Y. Siret, and E. Tournier. Computer Algebra (2nd ed.). *Academic Press*, 1993.

[Hea79] A.C. Hearn. Non-Modular Computation of Polynomial Gcd Using Trial Division. In *Proceedings EUROSAM 79*, pages 227–239, 1979.

[Kau08] M. Kauers. Integration of Algebraic Functions: A Simple Heuristic for Finding the Logarithmic Part. In D.J.Jeffrey, editor, *Proceedings ISSAC 2008*, pages 133–140, 2008.

[MN81] P.M.A. Moore and A.C. Norman. Implementing a Polynomial Factorization and GCD Package. In *Proceedings SYMSAC 81*, pages 109–116, 1981.

[Mus78] D.R. Musser. On the efficiency of a polynomial irreducibility test. *J. ACM*, 25:271–282, 1978.

[NM77] A.C. Norman and P.M.A. Moore. Implementing the New Risch Integration Algorithm. In *Proceedings 4th. Int. Colloquium on Advanced Computing Methods in Theoretical Physics*, pages 99–110, 1977.

[Ris79] R.H. Risch. Algebraic Properties of the Elementary Functions of Analysis. *Amer. J. Math.*, 101:743–759, 1979.

[vH02] M. van Hoeij. Factoring polynomials and the knapsack problem. *J. Number Theory*, 95:167–189, 2002.

[Wan78] P.S. Wang. An Improved Multivariable Polynomial Factorising Algorithm. *Math. Comp.*, 32:1215–1231, 1978.